Chapter 2

JUNOScript Session Control

This chapter explains how to start and terminate a session with the JUNOScript server, and describes the Extensible Markup Language (XML) tags that client applications and the JUNOScript server use to coordinate information exchange during the session. It discusses the following topics:

General JUNOScript Conventions

A client application must comply with XML and JUNOScript conventions. Compliant applications are easier to maintain if the JUNOS Internet software or the JUNOScript application programming interface (API) changes. The JUNOScript server always obeys the conventions. The following sections describe JUNOScript conventions:

Ordering and Context for Session Control Tags

A *session control* tag is one that delimits the parts of a JUNOScript session. There are tags that indicate the start or end of a session, identify a client request or server response, and signal error conditions. Most session control tags can occur only in certain contexts and in a prescribed order. JUNOScript session controllers include the following:

<?xml?>—An XML processing instruction (PI), emitted by both the client application and the JUNOScript server as they establish a JUNOScript session. For more information about PIs, see "XML Processing Instructions" on page 12.

<junoscript>—The root tag for every JUNOScript session, emitted by both the client application and the JUNOScript server as they establish and close the session.

<rpc>—The container tag that encloses each request emitted by the client application. It can occur only within <junoscript> tags.

<rpc-reply>—The container tag that encloses each response returned by the JUNOScript server. It can occur only within <junoscript> tags.

For more information about how to use these tags, see "Start, Control, and End a JUNOScript Session" on page 14 and the summary of session control tags in the *JUNOScript API Reference*.

Ordering and Context for Request and Response Tags

A *request* tag is one generated by a client application to request information about a router's current status or configuration or to change the configuration. A request tag corresponds to a JUNOS command-line interface (CLI) operational command or configuration statement. It can occur only within <rpc> session control tags.

A *response* tag represents the JUNOScript server's reply to a request tag and occurs only within <rpc-reply> tags.

The following example represents an exchange in which a client application emits the <get-interface-information> request tag with the <extensive/> flag and the JUNOScript server returns the <interface-information> response tag. (For information about the xmlns attribute, see "JUNOScript Server Response Classes" on page 27.)

Client Application

```
<rpc>
   <get-interface-information>
      <extensive/>
   </get-interface-information>
</rpc>
```

JUNOScript Server

```
<rpc-reply>
   <interface-information xmlns="URL">
      <!- - child tags of the <interface-information> tag - ->
   </interface-information>
</rpc-reply>
```

T1000

A client application can send only one request tag at a time to a particular router, and must not send another request tag until it receives the closing </rpc-reply> tag that represents the end of the JUNOScript server's response to the current request.

Ordering and Context for a Request Tag's Child Tags

Some request tags contain child tags. For configuration request tags, each child tag represents a level in the JUNOS configuration hierarchy. For operational request tags, each child tag represents one of the options you provide on the command line when issuing the equivalent CLI command.

Some request tags require that certain child tags be present. To make a request successfully, a client application must emit the required child tags within the request tag's opening and closing tags. If any of the request tag's children are themselves container tags, the opening tag for each must occur before any of the tags it contains, and the closing tag must occur before the opening tag for another tag element at its hierarchy level.

In most cases, the ordering of child tags at one hierarchy level within a request tag is not significant. The important exception is the *identifier tag* for an element, which distinguishes the element from other elements of its type. It must occur first within the container tag that represents the element. Most often the identifier tag specifies the element name and is called <name>. For more information, see "Tag Mappings for Identifiers" on page 42.

Ordering and Context for a Response Tag's Child Tags

A response tag's child tags represent the individual data items returned by the JUNOScript server for a particular request. At one hierarchy level within a response tag, there is no prescribed order for the child tags, and the set of child tags is subject to change in later releases of the JUNOScript API. Client applications must not rely on the presence or absence of a particular tag in the JUNOScript server's output, nor on the ordering of child tags within a response tag. For the most robust operation, include logic in the client application that handles the absence of expected tags or the presence of unexpected ones as gracefully as possible.

Spaces, Newlines, and Other White Space Characters

The JUNOScript API complies with the XML specification in ignoring spaces, newlines, and other characters that represent white space. Client applications must not depend on leading, trailing, or embedded white space when parsing the tag stream emitted by the JUNOScript server. For more information about white space in XML documents, see the XML specification at http://www.w3.org/TR/REC-xml.

Comments

Comments can appear at any point in the tag stream emitted by the JUNOScript server. Client applications must handle them gracefully but must not depend on their content. Client applications also cannot use comments to convey information to the JUNOScript server, because the server automatically discards any comments it receives.

Because the JUNOScript API is based on XML, comments are enclosed within the strings <!-- and -->, and cannot contain the string -- (two hyphens). For more details about comments, see the XML specification at http://www.w3.org/TR/REC-xml.

The following is an example of an XML comment:

```
<!-- This is a comment. Please ignore it. -->
```

XML Processing Instructions

An XML PI contains information relevant to a particular protocol and has the following form:

```
<?PI-name attributes?>
```

Some PIs emitted during a JUNOScript session include information that a client application needs for correct operation. A prominent example is the <?xml?> tag, which the client application and JUNOScript server each emit at the beginning of every JUNOScript session to specify which version of XML and which character encoding scheme they are using. For more information, see “Emit Initialization PIs and Tags” on page 21.

The JUNOScript server can also emit PIs that the client application does not need to interpret (for example, PIs intended for the JUNOS CLI). If the client application does not understand a PI, it must treat the PI like a comment rather than exiting or generating an error message.

Predefined Entity References

By XML convention, there are two contexts in which certain characters cannot appear in their regular form:

- In the string that appears between the opening and closing parts of a tag element (the value contained by the tag)
- In the string value assigned to an attribute of an opening tag

When including a disallowed character in either context, client applications must substitute the equivalent *predefined entity reference*, which is a string of characters that represents the disallowed character. The JUNOScript server uses the same predefined entity references in its response tags, so the client application must be able to convert them to actual characters when processing response tags.

Table 2 summarizes the mapping between disallowed characters and predefined entity references for strings that appear between the opening and closing tags of a tag element.

Table 2: Predefined Entity Reference Substitutions for Tag Content Values

Disallowed Character	Predefined Entity Reference
&	&amp;
<	&lt;
>	&gt;

Table 3 summarizes the mapping between disallowed characters and predefined entity references for attribute values.

Table 3: Predefined Entity Reference Substitutions for Attribute Values

Disallowed Character	Predefined Entity Reference
&	&amp;
'	&apos;
>	&gt;
<	&lt;
"	&quot;

As an example, suppose that the following string is the value contained by the <condition> tag:

```
if (a<b && b>c) return "Peer's dead"
```

Using the required predefined entity references, the <condition> tag looks like this:

```
<condition>if (a&lt;b &amp;&amp; b&gt;c) return "Peer's dead"</condition>
```

Similarly, if the value for the <example> tag's heading attribute is Peer's "age" <> 40, the opening tag looks like this when the required predefined entity references are used:

```
<example heading="Peer&apos;s &quot;age&quot; &lt;&gt; 40">
```

Start, Control, and End a JUNOScript Session

The JUNOScript server communicates with client applications within the context of a JUNOScript *session*. The server and client explicitly establish a connection and session before exchanging data, and close the session and connection when they are finished. The streams of JUNOScript tags emitted by the JUNOScript server and a client application must each constitute a well-formed XML document by obeying the structural rules defined in the JUNOScript document type definition (DTD) for the kind of information they are exchanging. The client application must emit tags in the required order and only in the allowed contexts.

Client applications access the JUNOScript server using one of the protocols listed in "Supported Access Protocols" on page 14. To authenticate with the JUNOScript server, they use either a JUNOScript-specific mechanism or the protocol's standard authentication mechanism, depending on the protocol. After authentication, the JUNOScript server uses the JUNOS login accounts and classes already configured on the router to determine whether a client application is authorized to make each request.

See the following sections for information about establishing, using, and terminating a connection and JUNOScript session:

For an example of a complete JUNOScript session, see "Example of a JUNOScript Session" on page 31.

Supported Access Protocols

The JUNOScript server accepts connections created using the access protocols listed in Table 4, which also specifies the associated authentication mechanism.

Table 4: Supported Access Protocols and Authentication Mechanisms

Access Protocol	Authentication Mechanism
clear-text, a JUNOScript-specific protocol for sending unencrypted text over a Transmission Control Protocol (TCP) connection	JUNOScript-specific
ssh (secure shell)	Standard ssh
SSL (Secure Sockets Layer)	JUNOScript-specific
telnet	Standard telnet

The SSL and ssh protocols are preferred because they encrypt security information (such as a password) before transmitting it across the network. The clear-text and telnet protocols do not encrypt security information.

For information about each protocol's authentication prerequisites, see "Prerequisites for Establishing a Connection" on page 15. For authentication instructions, see "Connect to the JUNOScript Server" on page 18.

Prerequisites for Establishing a Connection

Both the JUNOScript server and the client application must be able to access the software for the access protocol that the client application uses to create a connection. The JUNOScript server can access the protocols listed in "Supported Access Protocols" on page 14, because the JUNOS Internet software distribution includes them. On most operating systems, client applications can access the software for TCP (used by the JUNOScript-specific clear-text protocol) and the telnet protocol as part of the standard distribution. For information about obtaining ssh software for use by a client application, see http://www.ssh.com and http://www.openssh.com. For information about obtaining SSL software, see http://www.openssl.org.

For information about connection prerequisites, see the following sections:

Prerequisites for clear-text Connections on page 15

Prerequisites for ssh Connections on page 16

Prerequisites for SSL Connections on page 16

Prerequisites for telnet Connections on page 17

Prerequisites for clear-text Connections

If the client application uses the clear-text protocol to send unencrypted text directly over a TCP connection without using any additional protocol (such as ssh, SSL, or telnet), perform the following procedure to activate the xnm-clear-text service on the JUNOScript server machine. The service listens on port 3221:

1. Enter CLI configuration mode on the JUNOScript server machine and issue the following command:

   ```
   [edit]
   user@host# activate system services xnm-clear-text
   ```

2. Commit the configuration:

   ```
   [edit]
   user@host# commit
   ```

Prerequisites for ssh Connections

The ssh protocol uses public-private key technology. The ssh client software must be installed on the machine where the client application runs. If the ssh private key is encrypted (as is recommended for greater security), the ssh client must be able to access the passphrase used to decrypt the key.

If the client application uses the JUNOScript Perl module described in “Write a Perl Client Application” on page 71, no further action is necessary. As part of the Perl module installation procedure, you install a prerequisites package that includes the necessary ssh software.

If the client application does not use the JUNOScript Perl module, perform the following procedures to enable it to establish ssh connections:

1. Install the ssh client on the machine where the client application runs.

2. If the private key is encrypted (as recommended), use one of the following methods to make the associated passphrase available to the ssh client:

   Run the ssh-agent program to provide key management.

   Direct the ssh client to the file on the local disk that stores the passphrase.

   Include code in the client application that prompts a human user for the passphrase.

Prerequisites for SSL Connections

The SSL protocol uses public-private key technology, which requires a paired private key and authentication certificate. Perform the following procedure to enable a client application to establish SSL connections:

1. Install the SSL client on the machine where the client application runs.

   (Skip this step if the client application uses the JUNOScript Perl module described in “Write a Perl Client Application” on page 71. As part of the Perl module installation procedure, you install a prerequisites package that includes the necessary SSL software.)

2. Obtain an authentication certificate in Privacy Enhanced Mail (PEM) format, in one of two ways:

   Request a certificate from a Certificate Authority; these agencies usually charge a fee.

   Issue the following openssl command to generate a self-signed certificate; for information about obtaining the openssl software, see http://www.openssl.org.

   The command writes the certificate and an unencrypted 1024-bit RSA private key to the file called *certificate-file*.pem. The command appears here on two lines only for legibility:

   ```
   % openssl req -x509 -nodes -newkey rsa:1024 -keyout certificate-file.pem \
           -out certificate-file.pem
   ```

3. Enter CLI configuration mode on the JUNOScript server machine and issue the following commands to import the certificate. In the first command, substitute the desired certificate name for the *certificate-name* variable. In the second command, for the *URL-or-path* variable substitute the name of the file that contains the paired certificate and private key, either as a URL or a pathname on the local disk:

   ```
   [edit]
   user@host# edit security certificates local certificate-name

   [edit security certificates local certificate-name]
   user@host# set load-key-file URL-or-path
   ```

   

   The CLI expects the private key in the specified file (*URL-or-path*) to be unencrypted. If the key is encrypted, the CLI prompts for the passphrase associated with it, decrypts it, and stores the unencrypted version.

4. Issue the following commands to activate the xnm-ssl service, which listens on port 3220. In the last command, substitute the same value for the *certificate-name* variable as in Step 3:

   ```
   [edit security certificates local certificate-name]
   user@host# top

   [edit]
   user@host# edit system services

   [edit system services]
   user@host# activate xnm-ssl

   [edit system services]
   user@host# set xnm-ssl local-certificate certificate-name
   ```

5. Commit the configuration:

   ```
   [edit system services]
   user@host# commit
   ```

Prerequisites for telnet Connections

There are no prerequisites for enabling a client application to establish telnet connections, other than ensuring that both the client application and the JUNOScript server can access the telnet software. For a discussion, see “Prerequisites for Establishing a Connection” on page 15.

Connect to the JUNOScript Server

A client application written in Perl can most efficiently establish a connection and open a JUNOScript session by using the JUNOScript Perl module described in "Write a Perl Client Application" on page 71. For more information, see that chapter.

For a client application that does not use the JUNOScript Perl module, first perform the prerequisite procedures for the access protocol being used, as described in "Prerequisites for Establishing a Connection" on page 15. The supported access protocols are listed in "Supported Access Protocols" on page 14.

The client application must then perform the following steps:

1. Open a socket or other communications channel to the JUNOScript server machine (router) by invoking one of the remote-connection routines appropriate for the combination of programming language and access protocol that the application uses.

2. Provide authentication information as required by the access protocol:

   If using the clear-text or SSL protocol, authenticate with the JUNOScript server. For instructions, see "Authenticate with the JUNOScript Server" on page 19.

   Note that you must already have activated the authentication software on the JUNOScript server machine as specified in "Prerequisites for clear-text Connections" on page 15 and "Prerequisites for SSL Connections" on page 16.

   If using the ssh or telnet protocol, use the protocol's built-in authentication mechanism. Then issue the junoscript command to request that the JUNOScript server convert the connection into a JUNOScript session. For a C-code example, see "Write a C Client Application" on page 77.

3. Emit initialization tags as described in "Start the JUNOScript Session" on page 21.

For more information about authentication and establishing a connection, see the following sections:

Authenticate with the JUNOScript Server on page 19

Connect to the JUNOScript Server from the CLI on page 20

Authenticate with the JUNOScript Server

A client application that uses the clear-text or SSL protocol must authenticate with the JUNOScript server. The client application begins by emitting the <request-login> tag within <rpc> tags. In the <request-login> tag, it encloses the <username> tag to specify the name of the JUNOS user account under which to establish the connection. The account must already exist on the JUNOScript server machine. The application can choose whether to specify the password for the account as part of the current tag sequence:

To specify the password along with the JUNOS account name, emit the following tag sequence:

```
<rpc>
   <request-login>
      <username>JUNOS-account</username>
      <challenge-response>password</challenge-response>
   </request-login>
</rpc>
```

This tag sequence is appropriate if the application automates access to router information and does not interact with human users, or obtains the password from a human user before beginning the authentication process.

To specify only the JUNOS account name, emit the following tag sequence:

```
<rpc>
   <request-login>
      <username>JUNOS-account</username>
   </request-login>
</rpc>
```

This tag sequence is appropriate if the application does not obtain the password until the authentication process has already begun. In this case, the JUNOScript server returns the <challenge> tag within <rpc-reply> tags to request the password associated with the account. The tag encloses the string Password: which the client application can forward to the screen as a prompt for a human user. The echo attribute on the <challenge> tag is set to the value no to specify that the password string typed by the user does not echo on the screen. The tag sequence is as follows:

```
<rpc-reply>
   <challenge echo="no">Password:</challenge>
</rpc-reply>
```

The client application obtains the password and emits the following tag sequence to forward it to the JUNOScript server:

```
<rpc>
   <request-login>
      <username>JUNOS-account</username>
      <challenge-response>password</challenge-response>
   </request-login>
</rpc>
```

After it receives the account name and password, the JUNOScript server emits the <authentication-response> tag to indicate whether the authentication attempt is successful:

If the password is correct, the authentication attempt succeeds and the JUNOScript server emits the following tag sequence:

```
<rpc-reply>
  <authentication-response>
    <status>success</status>
    <message>JUNOS-account</message>
  </authentication-response>
</rpc-reply>
```

The *JUNOS-account* is the JUNOS account name under which the connection is established. The JUNOScript session begins as described in "Start the JUNOScript Session" on page 21.

If the password is not correct or the <request-login> tag stream is otherwise malformed, the authentication attempt fails and the JUNOScript server emits the following tag sequence:

```
<rpc-reply>
  <authentication-response>
    <status>fail</status>
    <message>error-message</message>
  </authentication-response>
</rpc-reply>
```

The *error-message* is a string explaining why the authentication attempt failed. The JUNOScript server emits the <challenge> tag up to two more times before rejecting the authentication attempt and closing the connection.

Connect to the JUNOScript Server from the CLI

The JUNOScript API is primarily intended for use by client applications, but for testing purposes you can establish an interactive JUNOScript session and type commands in a shell window. To connect to the JUNOScript server from JUNOS CLI operational mode, issue the junoscript command:

```
cli> junoscript
```

To begin a JUNOScript session over the connection, emit the initialization tags described in "Start the JUNOScript Session" on page 21. You can then type sequences of tags that represent operational and configuration operations, as described in "Operational Requests" on page 35 and "Router Configuration" on page 39. To eliminate typing errors, save complete tag sequences in a file and use a cut-and-paste utility to copy the sequences to the shell window.


Note

When you close the connection to the JUNOScript server (for example, by emitting the <request-end-session> and </junoscript> tags), the router completely closes your connection instead of returning you to the CLI operational mode prompt.

Similarly, the JUNOScript server completely closes your connection if there are any typographical or syntax errors in the tag sequence you enter.

Start the JUNOScript Session

Each JUNOScript session begins with a handshake in which the JUNOScript server and the client application specify the versions of XML and the JUNOScript API they are using. Each party parses the version information emitted by the other, using it to determine whether they can communicate successfully. The following sections describe how to start a JUNOScript session:

Emit Initialization PIs and Tags

When the JUNOScript session begins, the client application emits an <?xml?> PI and an opening <junoscript> tag, as described in the following sections.

Emit the <?xml?> PI

The client application begins by emitting an <?xml?> PI with the following syntax:

```
<?xml version="version" encoding="encoding"?>
```

The PI attributes are as follows. For a list of the attribute values that are acceptable in the current version of the JUNOScript API, see “Supported Protocol Versions” on page 25.

version—The version of XML with which tags emitted by the client application comply

encoding—The standardized character set that the client application uses and can understand

In the following example of a client application’s <?xml?> PI, the version="1.0" attribute indicates that it is emitting tags that comply with the XML 1.0 specification. The encoding="us-ascii" attribute indicates that the client application is using the 7-bit ASCII character set standardized by the American National Standards Institute (ANSI). For more information about ANSI standards, see http://www.ansi.org.

```
<?xml version="1.0" encoding="us-ascii"?>
```

Emit the Opening <junoscript> Tag

The client application then emits its opening <junoscript> tag, which has the following syntax:

```
<junoscript version="version" hostname="hostname" release="release-code">
```

The tag attributes are as follows. The version attribute is required, but the other attributes are optional. For a list of the attribute values that are acceptable in the current version of the JUNOScript API, see “Supported Protocol Versions” on page 25.

version—(Required) The version of the JUNOScript API that the client application is using.

hostname—The name of the machine on which the client application is running. The information is used only when diagnosing problems. The JUNOScript API does not include support for establishing trusted-host relationships or otherwise altering JUNOScript server behavior depending on the client hostname.

release—The identifier of the JUNOS Internet software release for which the client application is designed. It indicates that the client application can interoperate successfully with a JUNOScript server designed to understand that version of the JUNOS Internet software. In other words, it indicates that the client application emits request tags corresponding to supported features of the indicated JUNOS Internet software version, and knows how to parse response tags that correspond to those features. If you do not include this attribute, the JUNOScript server assumes that the client application can interoperate with its version of the JUNOS Internet software. For more information, see “Verify Compatibility” on page 24.

For *release-code*, use the standard notation for JUNOS Internet software version numbers. For example, the value 5.3R1 represents the initial version of JUNOS Release 5.3.

In the following example of a client application’s <junoscript> tag, the version="1.0" attribute indicates that it is using JUNOScript version 1.0. The hostname="client1" attribute indicates that the client application is running on the machine called client1. The release="5.3R1" attribute indicates that the router is running the initial version of JUNOS Release 5.3.

```
<junoscript version="1.0" hostname="client1" release="5.3R1">
```

Parse Initialization Tags from the JUNOScript Server

When the JUNOScript session begins, the JUNOScript server emits an <?xml?> PI and an opening <junoscript> tag, as described in the following sections.

Parse the JUNOScript Server’s <?xml?> PI

The syntax for the <?xml?> PI is as follows:

```
<?xml version="version" encoding="encoding"?>
```

The PI attributes are as follows. For a list of the attribute values that are acceptable in the current version of the JUNOScript API, see “Supported Protocol Versions” on page 25.

version—The version of XML with which tags emitted by the JUNOScript server comply

encoding—The standardized character set that the JUNOScript server uses and can understand

In the following example of a JUNOScript server's <?xml?> PI, the version="1.0" attribute indicates that the server is emitting tags that comply with the XML 1.0 specification. The encoding="us-ascii" attribute indicates that the server is using the 7-bit ASCII character set standardized by ANSI. For more information about ANSI standards, see http://www.ansi.org.

```
<?xml version="1.0" encoding="us-ascii"?>
```

Parse the JUNOScript Server's Opening <junoscript> Tag

The server then emits its opening <junoscript> tag, which has the following form (the tag appears on multiple lines only for legibility):

```
<junoscript version="version" hostname="hostname" os="JUNOS" release="release-code"
        xmlns="namespace-URL" xmlns:junos="namespace-URL"
        xmlns:xnm="namespace-URL">
```

The tag attributes are as follows.

version—The version of the JUNOScript API that the JUNOScript server is using.

hostname—The name of the router on which the JUNOScript server is running.

os—The operating system of the router on which the JUNOScript server is running. The value is always JUNOS.

release—The identifier for the version of the JUNOS Internet software from which the JUNOScript server is derived and is designed to understand. It is presumably in use on the router where the JUNOScript server is running. The *release-code* uses the standard notation for JUNOS Internet software version numbers. For example, the value 5.3R1 represents the initial version of JUNOS Release 5.3.

xmlns—The XML namespace for the tags enclosed by the <junoscript> tag that do not have a prefix on their names (that is, the default namespace for JUNOScript tags). The value is a URL of the form http://xml.juniper.net/xnm/*version*/xnm, where *version* is a string such as 1.1.

xmlns:junos—The XML namespace for the tags enclosed by the <junoscript> tag that have the junos: prefix on their names. The value is a URL of the form http://xml.juniper.net/junos/*release-code*/junos, where *release-code* is the standard string that represents a release of the JUNOS software, such as 5.3R1 for the initial release of version 5.3.

xmlns:xnm—The XML namespace for the JUNOScript tags enclosed by the <junoscript> tag that have the xnm: prefix on their names. The value is a URL of the form http://xml.juniper.net/xnm/*version*/xnm, where *version* is a string such as 1.1.

In the following example of a JUNOScript server's opening <junoscript> tag, the version attribute indicates that the server is using JUNOScript version 1.0 and the hostname attribute indicates that the router's name is big-router. The os and release attributes indicate that the router is running the initial version of JUNOS Release 5.3. The xmlns and xmlns:xnm attributes indicate that the default namespace for JUNOScript tags and the namespace for tags that have the xnm: prefix is http://xml.juniper.net/xnm/1.1/xnm. The xmlns:junos attribute indicates that the namespace for tags that have the junos: prefix is http://xml.juniper.net/junos/5.3R1/junos. The tag appears on multiple lines only for legibility.

```
<junoscript version="1.0" hostname="big-router" os="JUNOS" release="5.3R1"
        xmlns="http://xml.juniper.net/xnm/1.1/xnm"
        xmlns:junos="http://xml.juniper.net/junos/5.3R1/junos"
        xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm">
```

Verify Compatibility

Exchanging <?xml?> and <junoscript> tags enables a client application and the JUNOScript server to determine if they are running different versions of a protocol. Different versions are sometimes incompatible, and by JUNOScript convention the party running the later version of a protocol determines how to handle any incompatibility. For fully automated performance, include code in the client application that determines if its version of a protocol is later than that of the JUNOScript server. Decide which of the following options is appropriate when the application's version of a protocol is more recent, and implement the corresponding response:

Ignore the version difference, and do not alter standard behavior to accommodate the JUNOScript server's version. A version difference does not always imply incompatibility, so this is often a valid response.

Alter standard behavior to provide backward compatibility to the JUNOScript server. If the client application is running a later version of the JUNOS Internet software, for example, it can choose to emit only tags that represent the software features available in the JUNOScript server's version of the JUNOS Internet software.

End the JUNOScript session and terminate the connection. This is appropriate if you decide that accommodating the JUNOScript server's version of a protocol is not practical.

Supported Protocol Versions

Table 5 lists the protocol versions supported by version 1.0 of the JUNOScript API and specifies the PI or tag and attribute used to specify the information during JUNOScript session initialization.

Table 5: Supported Protocol Versions

Protocol and Versions	PI or Tag	Attribute
XML 1.0	<?xml?>	version="1.0"
ANSI-standardized 7-bit ASCII character set	<?xml?>	encoding="us-ascii"
JUNOScript 1.0	<junoscript>	version="1.0"
JUNOS Release 5.1 JUNOS Release 5.2 JUNOS Release 5.3 JUNOS Release 5.4	<junoscript>	release="5.1R*x*" release="5.2R*x*" release="5.3R*x*" release="5.4R*x*"

Exchange Tagged Data

The session continues when the client application sends a request to the JUNOScript server. The JUNOScript server does not emit any tags after session initialization, except in response to the client application's requests (or in the rare case that it needs to terminate the JUNOScript session). The following sections describe the exchange of tagged data:

Send a Request to the JUNOScript Server on page 25

Parse the JUNOScript Server Response on page 27

Send a Request to the JUNOScript Server

To initiate a request to the JUNOScript server, emit the opening <rpc> tag, followed by the tag or tags that represent a particular request, and the closing </rpc> tag, in that order. Enclose each request in a separate pair of opening <rpc> and closing </rpc> tags. For an example of emitting the <rpc> tag in the context of a complete JUNOScript session, see "Example of a JUNOScript Session" on page 31.

See the following sections for further information:

JUNOScript Request Classes on page 26

Include Attributes in the Opening < rpc > Tag on page 27

JUNOScript Request Classes

There are two classes of JUNOScript requests:

Operational requests—Requests for information about router status, which correspond to the JUNOS CLI commands listed in the *JUNOS Internet Software Operational Mode Command Reference*. The JUNOScript API defines a specific request tag for many CLI commands. For example, the <get-interface-information> tag corresponds to the show interfaces command, and the <get-chassis-inventory> tag requests the same information as the show chassis hardware command.

The following sample request is for detailed information about the interface called ge-2/3/0:

```
<rpc>
   <get-interface-information>
      <interface-name>ge-2/3/0</interface-name>
      <detail/>
   </get-interface-information>
</rpc>
```

For more information about requesting operational information, see "Operational Requests" on page 35. For a complete list of mappings between tags and CLI commands for the current version of the JUNOScript API, see the *JUNOScript API Reference*.

Configuration requests—Requests to change router configuration or for information about the current configuration, either candidate or committed (the one currently in active use on the router). The candidate and committed configurations diverge when there are uncommitted changes to the candidate configuration.

Configuration requests correspond to the JUNOS CLI configuration statements described in each of the JUNOS Internet software configuration guides. The JUNOScript API defines a tag for every container and leaf statement in the JUNOS configuration hierarchy.

The following example requests the information about the [edit system login] level of the current candidate configuration:

```
<rpc>
   <get-configuration>
      <configuration>
         <system>
            <login/>
         </system>
      </configuration>
   </get-configuration>
</rpc>
```

For more information about router configuration, see "Router Configuration" on page 39. For a summary of the available configuration tags, see the *JUNOScript API Reference*.

Although operational and configuration requests conceptually belong to separate classes, a JUNOScript session does not have distinct modes that correspond to CLI operational and configuration modes. Each request tag is enclosed within its own <rpc> tag, so a client application can freely alternate operational and configuration requests.

The client application can send only one request tag at a time to a particular router, and must not send another request tag until it receives the closing </rpc-reply> tag that represents the end of the JUNOScript server response to the current request.

Include Attributes in the Opening <rpc> Tag

Optionally, a client application can include one or more attributes in the opening <rpc> tag for each request. The client application can freely define attribute names. The JUNOScript server echoes each attribute, unchanged, in the opening <rpc-reply> tag in which it encloses its response. You can use this feature to associate requests and responses by defining an attribute in each opening request tag that assigns a unique identifier. The JUNOScript server echoes the attribute in its opening <rpc-reply> tag, making it easy to map the response to the initiating request.

Parse the JUNOScript Server Response

The JUNOScript server encloses its response to a client request in <rpc-reply> tags. Client applications must include code for parsing the stream of response tags coming from the JUNOScript server, either processing them as they arrive or storing them until the response is complete. See the following sections for further information:

JUNOScript Server Response Classes on page 27

Use a Standard API to Parse Response Tags on page 29

JUNOScript Server Response Classes

There are three classes of JUNOScript server responses:

Operational responses—Responses to requests for information about router status. These correspond to the output from JUNOS CLI operational commands as described in the *JUNOS Internet Software Operational Mode Command Reference*. The JUNOScript API defines response tags for all defined JUNOScript operational request tags.

For example, the JUNOScript server returns the information requested by the <get-interface-information> tag in a response tag called <interface-information>, and the information requested by the <get-chassis-inventory> tag in a response tag called <chassis-inventory>.

The opening tag for an operational response usually includes the xmlns attribute to define the XML namespace for the enclosed tags that do not have a prefix (such as junos:) before their names. The namespace is a URL of the form http://xml.juniper.net/junos/*release-code*/junos-*category*, where *release-code* is the standard string that represents the release of the JUNOS Internet software that is running on the JUNOScript server machine, and *category* represents the type of information.

The following sample response includes information about the interface called ge-2/3/0. The namespace indicated by the xmlns attribute contains interface information for the initial release of version 5.3 of the JUNOS Internet software. (Note that the opening <interface-information> tag appears on two lines only for legibility.)

```
<rpc-reply>
   <interface-information xmlns="http://xml.juniper.net/junos/5.3R1/junos-interface">
      <physical-interface>
         <name>ge-2/3/0</name>
         <!-- other data tags for the ge-2/3/0 interface -->
      </physical-interface>
   </interface-information>
</rpc-reply>
```

For more information about the contents of operational response tags, see “Operational Requests” on page 35. For a summary of operational response tags, see the *JUNOScript API Reference*.

Configuration information responses—Responses to requests for information about the router’s current configuration. The JUNOScript API defines a tag for every container and leaf statement in the JUNOS configuration hierarchy.

The following sample response includes the information at the [edit system login] level of the configuration hierarchy. For brevity, the sample shows only one user defined at this level.

```
<rpc-reply>
   <configuration>
      <system>
         <login>
            <user>
               <name>admin</name>
               <full-name>Administrator</full-name>
               <!-- other data tags for the admin user -->
            </user>
         </login>
      </system>
   </configuration>
</rpc-reply>
```

For more information about router configuration, see “Router Configuration” on page 39. For a summary of the available configuration tags, see the *JUNOScript API Reference*.

Configuration change responses— Responses to requests to change router configuration. In this case, the JUNOScript server indicates success by returning an opening <rpc-reply> and closing </rpc-reply> tag with nothing between them. It does not emit another tag that explicitly signals successful completion of the request.

For more information about router configuration, see "Router Configuration" on page 39. For a summary of the available configuration tags, see the *JUNOScript API Reference*.

For an example of parsing the <rpc-reply> tag in the context of a complete JUNOScript session, see "Example of a JUNOScript Session" on page 31.

Use a Standard API to Parse Response Tags

Client applications can handle incoming XML tags by feeding them to a parser that implements a standard API such as the Document Object Model (DOM) or Simple API for XML (SAX).

Routines in the DOM accept incoming XML and build a tag hierarchy in the client application's memory. There are also DOM routines for manipulating an existing hierarchy. DOM implementations are available for several programming languages, including C, C++, Perl, and Java. The DOM specification is available at http://www.w3.org/TR/REC-DOM-Level-1. Additional information is available at http://search.cpan.org/doc/TJMATHER/XML-DOM-1.37/lib/XML/DOM.pm (part of the Comprehensive Perl Archive Network [CPAN] Web site).

The version indicator in the URL for the DOM.pm file (1.37 in the preceding URL) is subject to frequent revision. Try substituting higher version numbers if the file cannot be found.

One potential drawback with DOM is that it always builds a hierarchy of tags and data, which can become very large. If a client application needs to handle only a subhierarchy at a time, it can use a parser that implements SAX instead. SAX accepts XML and feeds the tags and data directly to the client application, which must build its own tag hierarchy. For more information about SAX, see http://sax.sourceforge.net.

End the Session and Close the Connection

When a client application is finished making requests, it ends the JUNOScript session by emitting the empty <request-end-session/> tag within <rpc> tags. In response, the JUNOScript server emits the <end-session/> tag enclosed in <rpc-reply> tags. The client application waits to receive this reply before emitting its closing </junoscript> tag. For an example of the exchange of closing tags, see "Example of a JUNOScript Session" on page 31.

The client application can then close the ssh, SSL, or other connection to the JUNOScript server machine. Client applications written in Perl can close the JUNOScript session and connection by using the JUNOScript Perl module described in "Write a Perl Client Application" on page 71. For more information, see that chapter.

Client applications that do not use the JUNOScript Perl module use the routine provided for closing a connection in the standard library for their programming language.

Handle an Error Condition

If the JUNOScript server encounters an error condition that prevents it from processing the current request, it emits an <xnm:error> tag, which encloses child tags that describe the nature of the error. Client applications must be prepared to receive and handle an <xnm:error> tag at any time. The information in any response tags already received that are related to the current request might be incomplete. The client application can include logic for deciding whether to discard or retain the information.

An error can occur while the server is performing any of the following operations, and the server can send a different combination of child tags in each case:

- Processing a request submitted by a client application in a defined request tag
- Processing a command string submitted by a client application in a <command> tag (discussed in “Requests and Responses without Defined JUNOScript Tags” on page 37)
- Opening, locking, committing, or closing a configuration as requested by a client application (discussed in “Router Configuration” on page 39)
- Parsing a configuration file submitted by a client application in a <load-configuration> tag (discussed in “Change the Candidate Configuration” on page 55)

If the JUNOScript server encounters a less serious problem, it can emit an <xnm:warning> tag instead. The usual response for the client application in this case is to log the warning or pass it to the user, but to continue parsing the server’s response.

For a description of the child tags that can appear within an <xnm:error> or <xnm:warning> tag to specify the nature of the problem, see the entries for <xnm:error> and <xnm:warning> in the *JUNOScript API R eference*.

Halt a Request

To request that the JUNOScript server stop processing the current request, emit the empty <abort/> tag. The JUNOScript server responds with the empty <abort-acknowledgment/> tag. Depending on the operation being performed, response tags already sent by the JUNOScript server for the halted request are possibly invalid. The client application can include logic for deciding whether to discard or retain them as appropriate.

For more information about the <abort/> and <abort-acknowledgment/> tags, see their entries in the *JUNOScript API R eference*.

Display CLI Output as JUNOScript Tags

To display the output from a JUNOS CLI command as JUNOScript tags rather than the default formatted ASCII, pipe the command to the display xml command. The following example shows the output from the show chassis hardware command issued on an M40 Internet backbone router that is running the initial version of JUNOS Release 5.3:

```
user@host> show chassis hardware | display xml
<rpc-reply>
<chassis-inventory xmlns="http://xml.juniper.net/junos/5.3R1/junos-chassis">
<chassis junos:style="inventory">
<name>Chassis</name>
<serial-number>00118</serial-number>
<description>M40</description>
<chassis-module>
<name>Backplane</name>
<version>REV 06</version>
<part-number>710-000073</part-number>
<serial-number>AA2049</serial-number>
</chassis-module>
<chassis-module>
<name>Power Supply A</name>
...
</chassis-module>
...
</chassis>
</chassis-inventory>
</rpc-reply>
```

Example of a JUNOScript Session

This section describes the sequence of tags in a sample JUNOScript session. The client application begins by establishing a connection to a JUNOScript server.

The two parties then exchange initialization PIs and tags, as shown in the following example. Note that the JUNOScript server's <junoscript> tag appears on multiple lines for legibility only. The server does not actually insert a newline character into the list of attributes. For a detailed discussion of the <?xml?> PI and <junoscript> tag, see "Start the JUNOScript Session" on page 21.

Client Application

```
<?xml version="1.0" encoding="us-ascii"?>
<junoscript version="1.0" release="5.3R1">
```

JUNOScript Server

```
<?xml version="1.0" encoding="us-ascii"?>
<junoscript version="1.0" hostname="router1"
        os="JUNOS" release="5.3R1"
        xmlns="URL" xmlns:junos="URL"
        xmlns:xnm="URL">
```

T1001

The client application now emits the <get-chassis-inventory> tag to request information about the router's chassis hardware. The JUNOScript server returns the requested information in the <chassis-inventory> tag. In the following example, tags appear indented and on separate lines for legibility only. Client applications do not need to include newlines, spaces, or other white space characters in the tag stream they send to the JUNOScript server, because the server automatically discards such characters. Also, client applications can issue all tags that constitute a request within a single routine such as the C-language write() routine, or can invoke a separate routine for each tag or group of tags.

Client Application

```
<rpc>
   <get-chassis-inventory>
      <detail/>
   </get-chassis-inventory>
</rpc>
```

JUNOScript Server

```
<rpc-reply>
   <chassis-inventory xmlns="URL">
      <chassis>
         <name>Chassis</name>
         <serial-number>1122</serial-number>
         <description>M10</description>
         <chassis-module>
            <name>Midplane</name>
            <!- - other child tags for the Midplane chassis module - ->
         </chassis-module>
         <!- - tags for other chassis modules - ->
      </chassis>
   </chassis-inventory>
</rpc-reply>
```

T1002

The client application then prepares to create a new privilege class called network-mgmt at the [edit system login class] level of the configuration hierarchy. It begins by using the <lock-configuration/> tag to prevent any other users or applications from altering the candidate configuration at the same time. To confirm that the candidate configuration is locked, the JUNOScript server returns an <rpc-reply> and an </rpc-reply> tag with nothing between them.

Client Application

```
<rpc>
   <lock-configuration/>
</rpc>
```

JUNOScript Server

```
<rpc-reply></rpc-reply>
```

T1003

The client application emits the tags that define the new network-mgmt privilege class, commits them, and unlocks (and by implication closes) the configuration. As when it opens the configuration, the JUNOScript server confirms successful receipt, commitment, and closure of the configuration only by returning an opening <rpc-reply> and closing </rpc-reply> tag with nothing between them, not with a more explicit signal. (You do not need to understand the meaning of all tags at this point. For more information about configuring a router, see "Router Configuration" on page 39.)

Client Application	JUNOScript Server
<rpc>	
<load-configuration>	
<configuration>	
<system>	
<login>	
<class>	
<name>network-mgmt</name>	
<permissions>configure</permissions>	
<permissions>snmp</permissions>	
<permissions>system</permissions>	
</class>	
</login>	
</system>	
</configuration>	
</load-configuration>	
</rpc>	
	<rpc-reply></rpc-reply>
<rpc>	
<commit-configuration/>	
</rpc>	
	<rpc-reply></rpc-reply>
<rpc>	
<unlock-configuration/>	
</rpc>	
	<rpc-reply></rpc-reply>

T1004

The client application closes the JUNOScript session:

Client Application	JUNOScript Server
<rpc>	
<request-end-session/>	
</rpc>	
	<rpc-reply>
	<end-session/>
	</rpc-reply>
</junoscript>	
	</junoscript>

T1005